はめこみパズルのプログラム : TetrihexとTetraについて (計算機によるゲーム・パズルの具体化の検討)

AUTHOR(S):

川合, 慧

CITATION:

川合, 慧. はめこみパズルのプログラム : TetrihexとTetraについて (計算機によるゲーム・パズルの具体化の検討). 数理解析研究所講究録 1974, 217: 5-25

ISSUE DATE:

1974-07

URL:

http://hdl.handle.net/2433/105279

数理解析研究所講究録
217巻 1974年 5-25

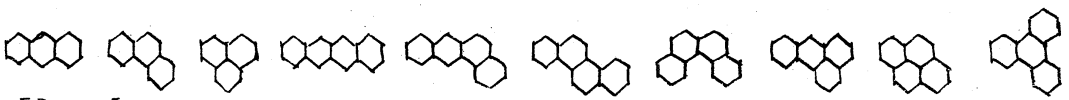
はめこみパズルのプログラム
— Tetrihex と Tetra について —

東大・理 情報科学 川合 慧

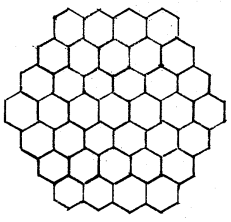
0. Tetrihex と Tetra

0.1 Tetrihex

互いに合同な正六角形を3個(Trihex)または4個(Tetrahex)つないだ図形を考える。回転や反転で移りかわれないものだけに限ると、つぎの10種類が得られる。([]内はpieceの名称)



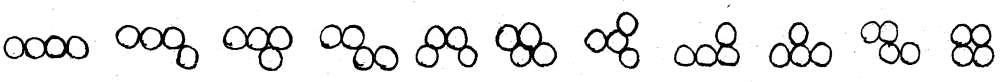
このpieceを全部使用して、一辺4の大きな正六角形状に単位正六角形を並べた図形を埋めるパズルをTetrahex — Trihex(略してTetrihexと呼ぶ)という。



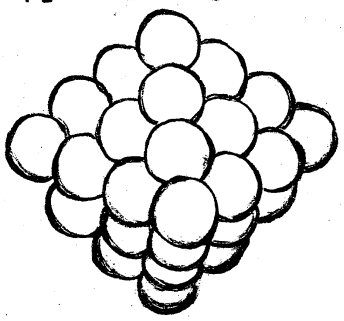
0.2 Tetra

互いに合同な球を4個、平面上でつないだものを考える。ただしつなぎ方はTetrominoまたはTetrahexと同じとする。回転や反転で移りかわれないものだけに限ると、つぎの11種

類が得られる。



[I] [J] [P] [S] [C] [Z] [Y] [L] [T] [N] [O]



このpieceを全部使用して，一辺の長さが4の正八面体状に球を積み上げた形にするパズルをTetraという。

0.3 解の総数

以上2つのパズルについて，幾何学的変換（回転や鏡映など）で移りかわれない解の総数は，

Tetrihex　12,290個

Tetra　7,482個　である。

1. はめこみパズル

1.1 はめこみパズルについて

世の中に存在している種々のパズルの中に，「はめこみパズル」というグループを形成しているものがある。ペントミノやペントキューブなどもこれに属する。これらのパズルに共通なものは，一定の形をした（平面的または立体的な）箱と，それを過不足なく埋めつくす小片の集合の存在である。そして，これらのパズルが面白い最大の理由は，各片の箱へのは

めこみ方の数は非常に多いのにもかかわらず、全片をはめこむことがかなり難しいことにある。この性質は、計算機を用いてパズルの解の総数を求めるという問題に対して、プログラミングの技法や哲学に関して少なからぬ意義を与えるものである。本文では、はめこみパズルに対して、その一般的な取扱いと解法、および実際の問題に適用した結果について述べ、さらにいくつかの考察を行なうことにする。

1.2 はめこみパズルの記述と定義

はめこみパズルの記述は、収容空間と各片およびその間の関係を規定することで一意的になされる。まず、抽象化した記述から始めることにする。

box とは、収容空間を形成している要素点である。通常のパズルでは、正方形や正六角形、立方体などの形をなしている。box に対しては、自然数（1,2,…）による一意的な名前づけを行なう。

frame とは、box の集合で片を収容する全体空間を指す。frame に含まれる box の総数を frame の体積という。

element とは、はめこみ操作の対象となる単位である。1つの element は1つの box に「はめこまれる」。

piece とは、一回のはめこみ操作で（いくつかの box にそれ

ぞれ）はめこまれるelementの集まりである。pieceは何種類かあり，異なるpieceに属する全element数は，frameの体積に等しい。

siteとは，pieceに属する各elementが，一度にはめこまれるboxの名前（番号）の組である。siteの数はpieceによって異なるのが普通である。

現実のパズルでは，siteに関するつぎのような性質が存在することが多い。

frame symmetryとは，boxに対する置換で，すべてのpieceのすべてのsiteが，それによって，同じpieceのsiteに移りかわるものの集合である。通常のパズルでは，二次元（三次元）的な回転や反転操作がこれにあたる。

local uniformityとは，boxからboxへの写像で，ある1つのpieceの（複数の）siteが，それによって，同じpieceのsiteに移りかわるものの集合である。平行移動や部分的な回転などが実際例としてあげられる。

adjacencyとは，boxに関して定義された2項関係で，任意のsiteについてその要素box を $b_1 \cdots b_n$，ただし b_i adjacent b_{i+1} $(i=1 \ldots n-1)$ と並べることのできるもののうち，極小のものを言う。adjacencyは，siteの表から一意的に定義できるとは限らない。通常のパズルにおいては，box相互の空間的な

隣接関係（の一部分）がこれに対応する。

これらの用語をもちいると，はめこみパズルの解をつぎのように定義することができる。

すべての piece から1つずつ選んだ site の組で，frame 中のすべての box が必ず1回だけその中にあらわれるもの。

2. はめこみパズルのプログラム

2.1 証明としてのプログラム

はめこみパズルの解の総数を求めるということは，「解の総数は xxxx である」という定理を証明することである。したがって，そのプログラム作成にあたっては，数学における証明問題を解くのと似た考慮を払わなければならない。ただし，数学においては「証明の過程」が書かれるのに対し，プログラムは単に「証明の方法」を示すのみであるという差異が存在する。したがって，正しい結果を得る（正しく証明する）ためには，「正しい」プログラムを書くことが必要である。それには，プログラムの構造を簡明にし，わかりやすいものにしなければならない。これは，ソフトウェア全般にわたって言えることがらであると思われるが，ここではこれ以上言及しない。

10

以上のことに関して，[0.3] に記した結果を得るために作成したプログラムにおいて試みたのは，つぎの各項目である。

(1) 単純な recursive call による簡明なプログラム構造

(2) 基礎データの機械的発生

(3) プログラム中断後の，再開アルゴリズムの定式化．

また，プログラム自体の効率を高めるため，つぎの工夫を行なった。

(4) 解の探索順序 .... most closely surrounded box 法.

(5) site 表の検索のための間接 sort 表の作成.

2.2 解の探索手順とプログラム構造

普通，はめこみパズルに関して採用される解の探索法は，固定された順番に従って piece を frame にはめこんでゆき，いきづまった時は一番最後に置いた piece の位置を少しずらして，またその先を試してゆくという方法である。これを，piece-search 法と呼ぼう。通常はこれを基本として，孤立点や分離部分の検出による無駄な探索の省略（feasibility test）を行なう。ここで [1.2] で述べた解の記述を見ると，各 piece が1回ずつ使用されるという言い方のほかに，各 box が1回ずつ含まれるという言い方も可能であることがわかる。そこで，ある局面における未使用 box の1つを選定し（これを，

target box という），未使用 piece の site のうち，その box を含むものをえらんではめこみ，いきづまったら一段戻るという方法が考えられる。これを box-search 法と呼ぶことにする。この方法は，一般的にいって piece-search 法よりも効率がよい。本プログラムも box-search 法を採用している。

＊プログラム構造　（以下の記述に関しては，本文末に掲げたプログラムを参照のこと）

プログラムの主な部分は search という部分である。search は，piece をいくつかはめこんだ状態をパラメータとして受け取り，その状態から出発したすべての解を求める。すべての解を求めるには，パラメータ状態における target box を決定し，その box を含むすべての未使用 piece の site についてはめこめるかどうかを調べ，はめこめる場合はその piece を"使用中"に，その site に含まれる各 box を"はめこまれ中"にしたうえで，自分自身を再び呼ぶ。プログラムにはこのほかに，すべての piece を使用したかどうかを調べる部分がついている。この recursive call の構造によって探索の方法が明確になると同時に，すべての場合を尽くしていることが保障される。

2.3 実際のプログラムについて

(1) target box　target box としては，自分自身が未使用

であり，かつそれに隣接している未使用 box の数が最小であるものを選ぶ。直感的には，最も入れにくい場所から埋めてゆくということになる。これを *most closely surrounded box* 法と名づけた。この *target box* を決定するために，つぎの補助変数を用意する。

(i) rempty (remained empty neighbours)： 各 box 毎に設けられ，自分のまわりの未使用 box 数を示す。使用されている box については，使われる直前の値の符号を変えたものが格納される。

(ii) group (same rempty value group)： rempty の値が同じである box の数を示す。rempty = 0 (孤立点を示す) から rempty の最大値 (max neighbours) に対応するものまで設けられている。

(2) 探索の順序　target box が決められたら，その box を含む site に関して，つぎの要領で探索を行なう。

(i) 未使用の piece を順番に全部調べる.

(ii) 1つの piece については，element を順番に全部調べる。

(iii) 1つの element については，それが target box にはめこまれる site を全部調べる。

この探索のために，つぎの3つの表を用意した。

(a) site の表．　三次元のデータ領域で，site[p, s, e] の内容が，p番目の piece の s番目の site の e番目の box 名をあらわす。ここで，p, s, e の最大値はそれぞれ number of piece, number of site[p], size of piece が示している。また，site の標準化のため

$$e_1 < e_2 \iff site[p, s, e_1] < site[p, s, e_2]$$

となっている。

(b) sort の表　やはり三次元のデータ領域で，site の表を element ごとに sort するために設けられた。

sort[p, s, e] = t であるということは，site[p, t, e] の値が，site[p, i, e] (i=1,2,…, number of site[p]) の中で s番目に小さいことを示している。すなわち，下の数列が単調増加列となる．

$$site[p, sort[p, s, e], e] \qquad (**)$$

ただし　$s = 1, 2, \cdots,$ number of site[p]

(c) box to site の表　上記 (**) の数列の中で，指定された box 名が始まる位置を示す三次元の表．具体的には

site[p, sort[p, box to site[p, b, e]-1, e], e] < b

site[p, sort[p, box to site[p, b, e], e], e] = b となる。box 名 b が (**) にあらわれないときは box to site[p, b, e] = 0 となる。

target box が与えられると，box to site → sort → site の順に表をたどることによって，その box を含むすべての site が，無駄なサーチをすることなしに求められる。

2.4 プログラム再開のためのアルゴリズム

組合せ問題用のプログラムは，とかく計算時間が長くなりがちである。その場合には，処理の中断・再開に対する考慮が必要である。二次記憶装置が使用できる場合には，作業領域の内容をすべて出力しておくことによって，プログラムの再開を行なうことができる。しかし，入出力時点での誤り（特に，人手が間に入る場合が問題）を減少させるためには，内部状態をできるだけ少数のパラメータで表現しておくことが望ましい。つぎに別の問題として，再開のためのアルゴリズムがあげられる。中断時点での内部状態の出力，再開時の入力，およびプログラム自体の再開など，どの段階に対しても完全な正しさが要求される。本プログラムでは，解の探索時とほとんど同じ手順を用いて，実行の再開を行なうアルゴリズムを採用した。したがって，探索そのものに対するのと全く同じ正しさが，再開アルゴリズムに対して保障されている。

もし，再開することだけが目的であって，それに要する時

間を問題にしないのであれば，中断時点での各boxの状態（未使用または特定のpieceがはめこまれている）を保持しておき，プログラムを先頭から走らせればよい。同じbox状態が現われたところで「再開作業」が終了するわけである。現実には，この方法は初めからやりなおすのと同じで、全く役に立たないが，これが本方式のヒントである。

本アルゴリズムでは，box状態の再現をpiecewiseに調べる。そのために，探索プログラムをつぎのように変更する。

(i) 初期化。探索モードの時は各boxを"未使用"とする．再開モードの時は，中断時点でのbox状態を読み込む。

(ii) はめこみテスト。探索モードでは、当該siteのboxがすべて未使用であるかどうかを調べる。再開モードでは，当該siteのすべてのboxに，指定されたpieceがはめこまれているかどうかを見る。

プログラムの中断点は，pieceの取り払いを行なう直前の場所である。解そのものも中断状態となる。

再開モードでは，探索順序に従ってpieceとそのsiteが決定されてゆく。そして，プログラムの中断点に達したとき，すべての制御変数が復元されて，再開モードから探索モードに切り換わる。［プログラム中のlocatableとpiece process参照］ このアルゴリズムは少し能率の悪い面もあるが，一

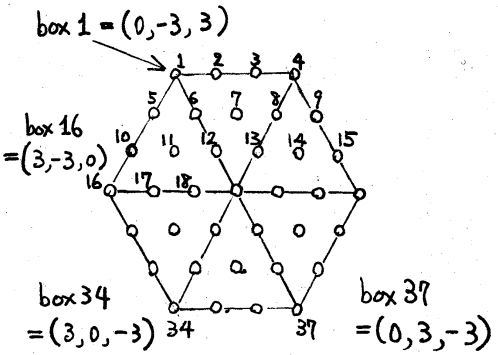
般的かつ明解なもので，他の問題に対しても充分応用が効くものである。

3. Site 表の作成

3.1 表の自動作成

はめこみパズルの site の表は，普通，かなり大きなものとなる。Tetra 程度のものでも site の数は 1,700 を越える。これを人手で作成した場合、誤りが皆無であることを主張するのはかなり困難である。したがって，基礎データである site 表も，なるべく少数のパラメータを用いて機械的に発生させることが望ましい。この場合には，各 box をあらわす座標系をうまくとって，各種の frame symmetry を利用することが重要となってくる。

3.2 Tetrihex の場合

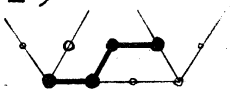


このパズルの frame symmetry は 60°回転対称と鏡映である。座標系は三次元で，各 box は X+Y+Z=0 の平面上に位置する。(右図参照) ある点の座標を (X_0, Y_0, Z_0) とすると，frame symmetry 操作によって得られる点の座標はつぎの通り。

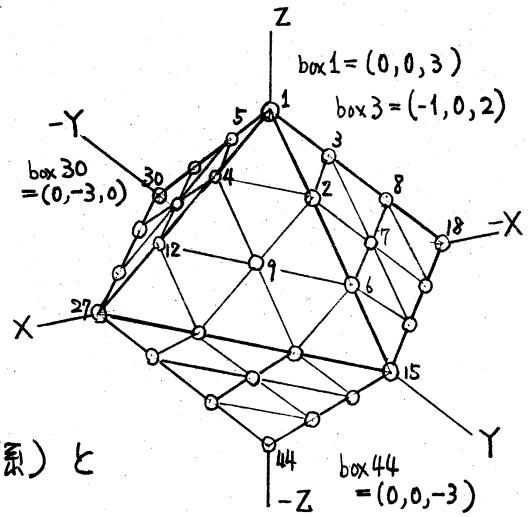
もとの点 (X_0, Y_0, Z_0)　　180°回転 $(-X_0, -Y_0, -Z_0)$
60°回転 $(-Y_0, -Z_0, -X_0)$　　240°回転 (Y_0, Z_0, X_0)
120°回転 (Z_0, X_0, Y_0)　　300°回転 $(-Z_0, -X_0, -Y_0)$
鏡映は，X座標値とY座標値を交換して得られる．

pieceのデータとしては，この座標系であらわしたsiteの1つを用意する。例えばpiece Sのデータはつぎのとうり。

$(3,0,-3), (2,1,-3), (1,1,-2), (0,2,-2)$



このデータを用いてpieceを平行移動させ，frame symmetry操作を行なったあと重複をとり除けばsite表が完成する。



3.3 Tetraの場合

このパズルでは座標系を右図のようにとった。frame symmetryによる座標値変化はつぎのとうり。ただし，回転軸は $X=\pm Y=\pm Z$（120°系）と $X=Y=0$（90°系）である。

もとの点 (X_0, Y_0, Z_0)　　90°　$(-Y_0, X_0, Z_0)$
120°　(Z_0, X_0, Y_0)　　90°+120°　$(-X_0, Z_0, Y_0)$
240°　(Y_0, Z_0, X_0)　　90°+240°　$(-Z_0, Y_0, X_0)$

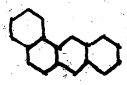
180°　$(-X_0, -Y_0, Z_0)$　　270°　$(Y_0, -X_0, Z_0)$
180°+120°　$(-Z_0, -X_0, Y_0)$　　270°+120°　$(X_0, -Z_0, Y_0)$
180°+240°　$(-Y_0, -Z_0, X_0)$　　270°+240°　$(Z_0, -Y_0, X_0)$

鏡映はZ座標値の符号をかえて得られる。

piece のデータと site 表の作成は，Tetrihex の場合と同様にして行なわれる。

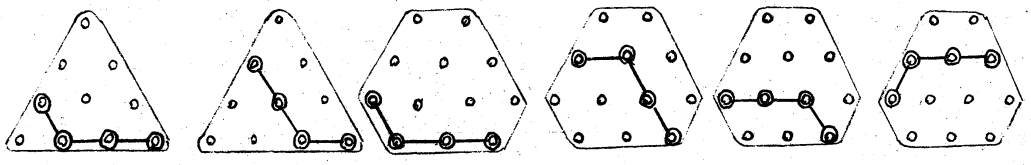
4. プログラム実行結果

4.1 Tetrihex

このパズルの解としては，frame symmetry で移りかわれないものだけを数えあげた。それには，適当な piece の site 表を制限すればよい。ここでは，piece J の site 表が frame symmetry の数だけのグループに分解できることを利用した。具体的には  の方向の site のみを許すことにすればよい。このような J の site は 18 あり，その他の piece の site は合計 911 である。使用したのは FACOM 270-20（加算 7.2 μs），言語は FORTRAN + ASSEMBLER である。印刷時間を除いて 5時間 40 分で，解の総数 12,290 が得られた。（この結果は，野下浩平氏によってすでに求められていたものと一致する）

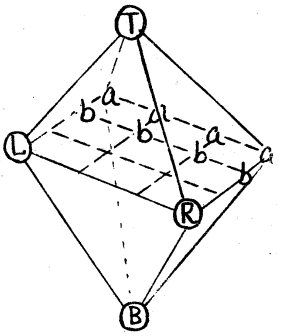
4.2 Tetra

このパズルに関しても，piece J を用いて frame symmetry をとり除いた。許される piece J の site はつぎの 6 つである。



左の2つは X+Y+Z=3の，残りは X+Y+Z=2 の平面上にある。その他の piece の solution は合計 1753 である。使用したのは，TOSBAC-40 (加算 5.6μs)，言語は ASSEMBLERである。印刷時間を入れて約 120 時間で，解の総数 7,482 が得られた。

このパズルについては，発案者の桑垣氏による解の標準化と分類法がある。概略はつぎのとうり。



(i) piece I を右図の a に入れたものを type 1，b に入れたものを type 2 とする。

(ii) a または b がのっている 4×4平面の，piece I から遠い方の2つの頂点を L, R とし，box L にはまっている piece 名が，box R のそれよりも alphabetical に小さくなるようにする。

(iii) 残りの頂点を T, B とし，(ii) と同じ標準化を行なう。

box T, B, L, R にはめこまれている piece 名をそれぞれ t, b, l, r とすると，任意の解の分類は tblr/n (n=1 or 2) となる。解全体をこの方式で分類したものを本文末に掲げる。

5. その他の考察

5.1 Feasibility Test

本プログラムでは，piece をはめこんだ時に group[0] ≠ 0

となったら，それ以上深い探索は行なわない。これは孤立した未使用 box が発生しているためである。この例のように，ある部分の探索が無駄である（解が存在し得ない）ことを検出する機構を Feasibility Test と呼ぶ。ほかに考えられるものとしては，未使用 piece の site がすべて使用不可能であることを検出するもの，孤立（連結）空間の発生を検出するものなどがある。前者の方法を，Tetra において実験した。piece としては I（site数 24）を用いたが，Test の有無は探索速度にほとんど影響しないという結果が得られた。これは，piece の site 使用不可の検査の手間が，その piece に関する探索の手間とほとんど同じであることを推測させる。

5.2 box search 法の効率について

2.2 でふれた piece search 法と box search 法について，簡単な「計算の手間」の見積りをおこなう。

piece 数を N，piece あたりの平均 site 数を S，frame の全体積を V，piece の大きさを M とする。n 個の piece がはめこまれた状態での一段分の探索は，

(i) site を探して frame にはめこめるかテストする（計算の手間 A，回数 α_n），

(ii) piece をはめこみ，解を全部探した後に再び取り除く（

その段の計算の手間 B，回数 β_n ） となる。

これより，すべての手間はつぎの式であらわされる。

$$A\left(\alpha_0+\alpha_1\beta_0+\alpha_2\beta_1\beta_0+\cdots+\alpha_{N-1}\beta_{N-2}\cdots\beta_0\right)+B\left(\beta_0+\beta_1\beta_0+\cdots+\beta_{N-1}\cdots\beta_0\right)$$

(1) piece search の場合

$$\alpha_n = S\,,\quad \beta_n = S\left(1-\frac{n}{N}\right)^M$$

(2) box search (random selection) の場合

$$\alpha_n = S\cdot\frac{M}{V}\cdot(N-n)$$ （条件に合うsite数）×（未使用piece数）

$$= S\left(1-\frac{n}{N}\right)$$ （$NM=V$ と仮定している）

$$\beta_n = \alpha_n\left(1-\frac{n}{N}\right)^{M-1}$$

$$= S\left(1-\frac{n}{N}\right)^M$$

α_n の比較により，box search 法の方が明らかにすぐれていることがわかる。また，most closely surrounded box 法においては，target box 以外の $M-1$ 個の box が全部未使用である確率が $\left(1-\frac{n}{N}\right)^{M-1}$ よりかなり小さくなると予想されるので，計算の手間はかなり少なくなっていると思われる。Tetra において部分的に比較した様子では、piece search 法の手間と本プログラムの手間の比は 500 対 1 以上のようである。

6. まとめ

はめこみパズルに関して，ある程度の定式化を行なってみたが，解の正当性という問題も含めて，今後もっと考えてゆ

かなければならないと思っている。プログラムに関しては，frame symmetry の除去を自動的に行なうこと，効率のよい feasibility test を見つけることなどが，当面の問題として残されている。パズルのプログラムは，アルゴリズムやデータ構造などに関して，ソフトウェアのエッセンスを集めたような面を持っており，プログラミング教育の題材に適しているほかに，プログラミング言語や各種のアルゴリズム開発などにも有用なものであると思われる。

本プログラムの探索実行部分を付録として付けておく。言語は informal なものであるが，アルゴリズムの伝達には差しつかえないであろう。プログラム中で多用されている記号 ... は Algol 60 の step 1 until と思って読むこと。また，for 文の制御変数は，その文の先頭で整数型の変数として自動的に宣言され，その文の中でのみ有効である。

参考文献　E.W. Dijkstra「Notes on Structured Programming」in Structured Programming, Academic Press 1972.
(特に §17. The Problem of the Eight Queens)

```
I IL ILY ILNN
  SY OSY ONNC TLC
     JJJ POSY OTS TC
         PZZJ PZZ PC T
```

100th solution of the TETRA.

```
      J 3 3 3
    I J J J V
  Z I S V V D
Z Z I S S D D
  Z C I Y S P
    C Y Y P P
      C C Y P
```

100th solution of the TETRIHEX.

TYPE-I

T / B / RL	CCCCCCCCCJJJJJJJJLLLLLLLNNNNNCCCCPP-PSST JLNOPSTZLNOPSTZNOPSTZOPSTZPSTZSTZTZZ
CJ	..............0BCB5350000CDBAD14810
CL	.........0B7831.....0000026529321 12
CN	........0.40500.00210.....2AB3000120
CO	........D5.CEBB2.BCB4.29B1....BB5B65
CP	........90B.9610B.5214.200.192...220
CS	........D2CC.672C9.53A1.11B.C5.26..7
CT	........50A11.00554.1703.04A.63.1.2.
CZ	........101020.02100.0100.783.70.2..
JL	..0D7B70.............859 12DDBBEAAAB0
JN	.1.A2202........C5823.....CD66807271
JO	.DB.CDC2.......B.DEBB.3D74....DDBCC2
JP	.C5C.DB0.......1C.DB81.600.AC0...8A5
JS	.EAED.AC.......4FE.CDD9.4BE.EB.CD..B
JT	.51BC5.0.......1AB5.A044.0DB.AB.3.3.
JZ	.60B3D0........3CAD3.4142.998.D6.3..
LN	0..50300..25600..........0701700010
LO	B.2.CCC3.2.EE46.......2A15....C38BC2
LP	2.0B.411.17.DA6......2.000.633...282
LS	B.2BA.53.7DF.AD......C9.04C.C7.7D..0
LT	1.0511.0.03A6.2......011.0A5.32.3.0.
LZ	0.00410..05CB1.......0010.A76.B4.0..
NO	24..85A1A..CC12..2B10.........C452A0
NP	01.6.1002.4.600.1.100......920...110
NS	01.C2.113.AC.6B.D4.24.....C.9B.7A..0
NT	10.002.00.302.0.021.0.....72.12.0.0.
NZ	00.2030.0.0100..0030......430.20.0..
OP	996..BB7C1..DC54..C8B..654.......B97
OS	ECC.D.CDEC.F.CDD.D.DC.C.BB.....DC..C
OT	D22.68.1A2.BD.91.67.3.03.0....A.8.3.
OZ	492.8A6.C5.CC7.1.BB3..290.....98.8..
PS	9B3C..3AE1D..CB1C..CB5...40..DB.....3
PT	3409.4.000B.B.201.7.12.4.0.2.2....0.
PZ	0304.10.A14.B5.06.22.4.00..01....1..
ST	72043..0D9CD..20BC..28A..2B..3..2...
SZ	572AA.2.C3DE.8.3CA.4.86.0.C.B..2....
TZ	000402..00626..0451..110..43..3.....

TYPE-II

T / B / RL	CCCCCCCCCJJJJJJJJLLLLLLLNNNNNOOOOPPPSST JLNOPSTZLNOPSTZNOPSTZOPSTZPSTZSTZTZZ
CJ	0010000000000000000
CL	.........000000.....0000001200000 00
CN	........0.00000.00000.....1203000000
CO	........00.01030.0020.0062....020100
CP	........001.00000.1000.000.010...000
CS	........0040.00030.0000.000.30.00..0
CT	........00000.00000.0000.000.00.0.0.
CZ	........000010.00000.0000.000.01.0..
JL	..000000.............000000011000000
JN	.0.00000........B0000.....2010110010
JO	.00.0001.......0.0014.0000....020000
JP	.000.000.......00.0000.010.300...000
JS	.0000.10.......010.0000.000.50.10..0
JT	.00000.0.......0B01.0721.060.00.0.0.
JZ	.000001........02000.0000.002.20.2..
LN	0..00000..00010..........1000200000
LO	0.0.0000.0.0001.......0000....001100
LP	0.00.000.02.010......0.000.040...200
LS	0.000.00.050.00......10.002.13.00..0
LT	0.0000.0.0200.0......000.010.02.0.0.
LZ	0.01000..00000.......0000.111.00.0..
NO	01..10200..0241..0000.........000432
NP	00.0.0201.0.000.0.000......212...100
NS	00.00.000.00.00.20.00.....1.70.10..0
NT	00.530.00.200.0.000.0.....34.00.0.0.
NZ	00.1000.0.0000..0000......002.00.0..
OP	010..00000..3000..220..020.......060
OS	000.0.0001.1.000.1.01.1.23.....04..0
OT	000.10.000.20.10.31.0.00.0....1.0.1.
OZ	000.030.00.000.0.001..001.....00.0..
PS	0000..00020..0112..010..10..00.....1
PT	0000.0.0003.3.002.0.00.0.0.0.0....0.
PZ	0000.00.000.00.00.00.0.00..00....0..
ST	00000..00020..0090..0A1..03..0..0...
SZ	00000.0.0001.0.030.2.00.0.0.1..2....
TZ	000000..00000..0000..000..30..0.....

	TYPE-I		TYPE-II	
	TB	RL	TB	RL
CJ	156	208	0	1
CL	209	71	2	3
CN	80	46	0	6
CO	310	307	7	17
CP	203	82	5	3
CS	214	225	3	10
CT	154	68	6	0
CZ	84	35	1	2
JL	368	284	1	2
JN	61	150	3	22
JO	271	516	21	8
JP	602	238	4	4
JS	348	736	10	8
JT	156	201	6	31
JZ	158	223	7	9
LN	79	37	1	4
LO	364	303	54	3
LP	304	111	5	9
LS	274	385	5	12
LT	160	56	8	5
LZ	165	96	6	4
NO	128	155	18	20
NP	78	35	4	9
NS	117	196	1	11
NT	54	23	13	17
NZ	54	18	5	3
OP	370	245	24	16
OS	210	760	13	14
OT	261	163	32	10
OZ	129	193	9	5
PS	337	351	8	9
PT	173	70	10	8
PZ	198	63	5	0
ST	120	205	11	25
SZ	138	263	11	9
TZ	73	42	3	3

COUNT N = 0...9, A(10-12), B(13-19), C(20-29), D(30-49), E(50-99), F(100-) 7160 322

```
BEGIN {constants for TETRA}
  INT CONST number of piece=11, size of piece=4, number of box=44,
            max neighbours=12;
      {working spaces and tables}
  [1:number of piece, 1:FLEX, 1:size of piece]INT site,sort;
  [1:number of piece, 1:number of box, 1:size of piece]INT box to site;
  [1:number of box, 1:max neighbours]INT link;
  [0:max neighbours]INT group; [1:number of piece]INT number of site, BOOL used;
  [1:number of box]INT rempty,neighbour,box; BOOL set,delete,recovering;
      {procedure declarations}
  INT PROC most closely surrounded box;
    BEGIN INT nn,bb;
      FOR n=1... WHILE group[n]=0 DO nn := n;        {find group}
      FOR b=1... WHILE rempty[b]≠nn+1 DO bb := b;    {find box in the group}
      most closely surrounded box := bb+1
    END;
  BOOL PROC locatable(piece, site pointer); INT piece, site pointer;
    BEGIN INT target {is the desired state of boxes when searching or recovering};
      IF recovering THEN target := piece ELSE target := 0;
      locatable := TRUE;
      FOR i=1...size of piece WHILE locatable DO
                  locatable &:= box[site[piece, site pointer, i]]=target
    END;
  BOOL PROC all pieces used;
    BEGIN all pieces used := TRUE;
          FOR piece=1...number of piece WHILE all pieces used DO
                       all pieces used &:= used[piece]
    END;
  PROC piece process(function, piece, site pointer);
    BOOL function; INT piece, site pointer;
    BEGIN INT pbox,n;
      FOR i=1...size of piece DO  {each element operation}
        BEGIN pbox := site[piece, site pointer, i];
              FOR j=1...max neighbours WHILE link[pbox,j]>0 DO
                        neighbour[link[pbox,j]] +:= 1;
              n := rempty[pbox];  rempty[pbox] := -n;  group[ABS(n)] -:= 1;
              IF function=set THEN box[pbox] := piece ELSE box[pbox] := 0
        END ;
      FOR nbox=1...number of box DO  {empty box operation}
        IF neighbour[nbox]≠0 THEN  {this box must be processed}
```

```
        BEGIN IF rempty[nbox]>0 THEN {change the rempty of this box}
                BEGIN group[rempty[nbox]] -:= 1;
                      IF function=set THEN rempty[nbox] -:= neighbour[nbox]
                                      ELSE rempty[nbox] +:= neighbour[nbox];
                      group[rempty[nbox]] +:= 1
                END;
              neighbour[nbox] := 0
        END ;
    used[piece] := function;  recovering &:= function
  END;
PROC search;  {search main part}
  BEGIN INT target box, site pointer;
    target box := most closely surrounded box;
    FOR piece=1...number of piece DO  {test all pieces}
      IF ¬used[piece] THEN
        FOR element=1...size of piece DO  {test all elements}
          FOR bxst=box to site[piece, target box, element]...number of site[piece]
            WHILE site[piece,sort[piece,bxst,element],element]=target box DO
              BEGIN site pointer := sort[piece,bxst,element];  {test a site}
                    IF locatable(piece, site pointer) THEN
                      BEGIN piece process(set, piece, site pointer);
                            IF all pieces used THEN PRINT RESULT
                                              ELSE IF group[0]=0 THEN search;
                            piece process(delete, piece, site pointer)
                      END {of a site}
              END {of same element sites}
          {end of a piece}
  END;
    {initialize and table setting up (extrnal procedures)}
initialize(used,box,neighbour); set tables(link,site,sort,box to site,number of site);
set := TRUE;  delete :=FALSE;  {piece handling flag}
    {mode check (initial or recovering)}
READ IN(recovering);  IF recovering THEN READ STATUS INTO(box);
    {search main program start}
FOR first piece pointer=1...number of site[1] DO
  IF locatable({piece =}1, first piece pointer) THEN
    BEGIN piece process(set, 1, first piece pointer);
          search;
          piece process(delete, 1, first piece pointer)
    END {of search for a first piece site}
END  {of program}
```